Fast Iterative Solvers of SLE

- crucial drawback of solvers discussed so far: they become slower if we discretize more accurate!
- now: look for possible remedies
  - **relaxation**: explicit application of the *multigrid* principle
  - **Krylov/cg**: *preconditioning* (typically also following multigrid)
- let us start with preconditioning:
  - crucial quantity for cg's convergence: condition number
  - PDE: condition of system matrix increases dramatically with n
  - therefore: look for a modified matrix with better condition

$$Ax = b \Leftrightarrow M^{-1}Ax = M^{-1}b \Leftrightarrow W^{-1}AW^{-T}y = W^{-1}b, \quad \text{where}$$

$$M \text{ s.p.d.}, \quad WW^T = M, \quad y = W^T x, \quad M^{-1} \text{ and } W^{-1}AW^{-T} \text{ similar}$$

(no need to construct M or W explicitly, must be applied only)

Strategies for Preonditioners

- the simplest choice: $M = I$ (cheap, but useless)
- the best choice: $M = A$ (perfect, but expensive)
- some possibilities in-between:
  - **diagonal** or **Jacobi** preonditioner: $M = D_A$
  - GS or SOR are not used due to lack of symmetry
  - **SSOR** preconditioner: $M^{(1/2)} = \alpha^{-1}D_A + L_A; \quad M^{(1)} = \alpha^{-1}D_A + U_A;$

$$M = \frac{\alpha}{\alpha - 2}\left(M^{(1/2)}\right)^{-1} D_A^{-1} M^{(1)}$$

  - **incomplete factorization**, e.g. ILU: compute approximate factors L and U instead of exact ones in direct methods
  - **sparse approximate inverse**: look for some cheap B with

$$\min_B \|I - AB\|^2, \quad M^{-1} = B$$

  - **multilevel** preconditioners: following the multigrid principle

The Multigrid Principle

- starting point: *Fourier mode analysis* of the errors
  - decompose the error $e^{(i)} = x^{(i)} - x$ into its Fourier components (Fourier transform)
  - observe how they change/decrease under a standard relaxation like Jacobi or Gauß-Seidel (in a two-band sense):
    - The *high* frequency part (with respect to the underlying grid) is reduced quite quickly.
    - The *low* frequency part (w.r.t. the grid) decreases only very slowly; actually the slower, the finer the grid is.
  - This behaviour is annoying
    - the low frequencies are not expected to make troubles, but we can hardly get rid of them on a fine grid;

    but also encouraging
    - the low frequencies can be represented and, hopefully tackled, on a coarser grid – there is no need for the fine resolution.

A Simple Example

- 1D Laplace equation, u(0)=u(1)=0 (exact solution 0)
- equidistant grid, 65 points, 3-point stencil, damped Jacobi method with damping parameter 0.5
- start with random values in [0,1] for u in the grid points
- After 100 (!) steps, there is still a maximum error bigger than 0.1 due to low-frequency components!
- therefore the name *smoothers* for relaxation schemes:
  - They reduce the strongly oscillating parts of the error quite efficiently.
  - They, thus, produce a **smooth** error which is very resistent.
- the idea: work on grids of different resolution

Coarse Grid Correction 1

- sequence of equidistant grids on our domain:
  Ω_l, $l = 1,2,...,L$, with mesh width $h_l = 2^{-l}$
- let A_l, b_l, ... denote corresponding matrix, r.h.s., ...
- combine work on two grids with a *correction scheme* :

  smooth the current solution x_l;
  form the residual $r_l = b_l - A_l x_l$;
  restrict r_l to the coarse grid Ω_{l-1};
  provide a solution to $A_{l-1} e_{l-1} = r_{l-1}$;
  prolongate e_{l-1} to the fine grid Ω_l;
  add the resulting correction to x_l;
  if necessary, smooth again;

Coarse Grid Correction 2

- the different steps of this *2-grid algorithm* :
  - the **pre-smoothing**: reduce high-frequency error components, smooth error, and prepare residual for transfer to coarse grid
  - the **restriction**: transfer from fine grid to coarse grid
    - *injection* : inherit the coarse grid values and forget the others
    - *(full) weighting* : apply some averaging process
  - the **coarse grid correction**: provide an (approximate) solution on the coarse grid (direct, if coarse enough; some smoothing steps otherwise)
  - the **prolongation**: transfer from coarse grid to fine grid
    - usually some *interpolation* method
  - the **post-smoothing**: sometimes reasonable to avoid new high-frequency error components
- recursive application leads to multigrid methods

The V-Cycle

- now, the coarse grid equation is solved by coarse grid correction, too; the resulting algorithmic scheme is called *V-cycle* :

  smooth the current solution x_l;
  form the residual $r_l = b_l - A_l x_l$;
  restrict r_l to the coarse grid Ω_{l-1};
  solve $A_{l-1} e_{l-1} = r_{l-1}$ by coarse grid correction;
  prolongate e_{l-1} to the fine grid Ω_l;
  add the resulting correction to x_l;
  if necessary, smooth again;

- on the finest grid: direct solution
- number of smoothing steps: typically small (1 or 2)

Multigrid Algorithms

- the V-cycle is not the only multigrid scheme:
  - the **W-cycle**: after each prolongation, visit the coarse grid once more, before moving on to the next finer grid
  - the **nested iteration**: start on coarsest grid Ω_1 , smooth, prolongate to Ω_2 , smooth, prolongate to Ω_3 , and so on, until finest grid is reached; now start V-cycle
  - **full multigrid**: replace `smooth´ steps above by `apply a V-cycle´; combination of improved start solution and multigrid solver
- multigrid idea is not limited to rectangular or structured grids: we just need a hierarchy of nested grids (works for triangles or tetrahedra, too)
- also without underlying geometry: algebraic multigrid

Basic Convergence Results

- Cost (storage and computing time):
  - 1D: $c \cdot n + c \cdot n/2 + c \cdot n/4 + c \cdot n/8 + \ldots \leq 2c \cdot n = O(n)$
  - 2D: $c \cdot n + c \cdot n/4 + c \cdot n/16 + c \cdot n/64 + \ldots \leq 4/3 \cdot c \cdot n = O(n)$
  - 3D: $c \cdot n + c \cdot n/8 + c \cdot n/64 + c \cdot n/512 + \ldots \leq 8/7 \cdot c \cdot n = O(n)$
  - i.e.: work on coarse grids is negligible compared to finest grid
- Benefit (speed of convergence):
  - always significant acceleration compared with pure use of smoother (relaxation method)
  - in most cases even ideal behaviour $\gamma = O(1 - \text{const.})$
  - effect:
    - constant number of multigrid steps to obtain a given number of digits
    - overall computational work increases only linearly with *n*